TECHNICKÁ UNIVERSITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: Informační technologie

Studijní obor: Informační technologie

Zpracování a vizualizace rozsáhlé databáze měření kvality elektrické energie

Processing and visualisation of a large power quality dataset

Bakalářská práce

Autor: Jan Markgraf

Vedoucí práce: Ing. Jan Kraus

Konzultant: Ing. Pavel Štěpán

V Liberci 3.5. 2011

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č.121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická universita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiju-li bakalářkou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářkou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Abstrakt

Cílem této práce je vytvoření aplikace pro operační systém Windows v grafickém subsystému Windows Presentation Foundation, jejíž funkcí je vizualizace časových průběhů naměřených veličin získaných z různých měřících přístrojů. Data jsou do programu načítána z databáze za pomoci technologie eXpress Persistent Objects a aplikace využívá integrovaného 3D enginu Windows Presentation Foundation.

Klíčová slova

Windows Presentation Foundation, ENVIS, 3D engine, graf, vizualizace

Abstract

This paper focuses on creation of an application for the Windows operation system with the Windows Presentation Foundation graphic subsystem. The function of the application is to visualize measurements in time, obtained from various measuring devices. The data are loaded into the application from a database using eXpress Persistent Objects technology and the application uses Windows Presentation Foundation integrated 3D engine.

Keywords

Windows Presentation Foundation, ENVIS, 3D engine, chart, visualization

Typografické konvence

V textu jsou použity různé styly textu a formátování, které pomáhají rozlišit odlišné typy informací:

- Pro obyčejný text je použito běžné proporcionální patkové písmo.
- *Kurzívu* užívám v poznámkách, které obsahují doplňující informace k textu. Taktéž je užita v názvech ovládacích prvků aplikací apod.
- Celistvé úryvky zdrojových kódů a výrazy ze zdrojových kódů, které se vyskytují v textu, je zvykem označovat `neproporcionálním písmem`.

Obsah

1. Úvod

Tato práce se zabývá zpracováním velkého objemu dat měření elektrických veličin a následné interpretaci takových dat použitím vhodných metod vizualizace. Data jsou čerpána z databáze aplikace ENVIS firmy KMB Systems, která se zabývá vývojem průmyslových přístrojů pro měření kvality elektrické energie.

Výše zmíněná aplikace ENVIS je nástroj umožňující komunikaci mezi měřícími přístroji a osobním počítačem. Archivuje naměřená data v databázi SQL serveru nebo v CEA souborech. Tato data je poté možné v aplikaci analyzovat a vytvářet celou řadu různých vizualizací. Charakter aplikace a dostupných vizualizací může však pro méně zkušeného uživatele působit až příliš komplexně, zejména na velkém objemu dat.

Přichází tedy požadavek vytvořit poněkud „jednodušší“ pohledy na měřená data, zejména využít třetí rozměr. 3D vizualizace nejsou v oblasti měření a regulace elektrické energie obvyklé, proto bylo úkolem ověřit možnosti takových vizualizací a jejich praktickou využitelnost.

Za tímto účelem byla vytvořena modelová aplikace v prostředí Microsoft .NET Framework v jazyce C#, která spolupracuje s databází ENVIS. Aplikace umožňuje vybírat, jaká data se mají zpracovat, dále aplikovat na ně různé filtry, a výsledná data nakonec vizualizovat vybraným typem grafu.

Aplikace byla vytvořena v grafickém subsystému Windows Presentation Foundation (dále WPF). Fakt, že firma Microsoft zastavila další vývoj Windows Forms, dnes už zastaralého prostředí pro tvorbu desktopových aplikací pro Windows, je dostatečným důvodem k použití WPF, které je považováno za nástupce Windows Forms.

WPF navíc disponuje novým grafickým jádrem založeným na Direct3D, což programátorovi umožňuje využít vestavný 3D engine. To znamená, že není potřeba složitě integrovat nějaký externí 3D engine do WPF aplikace. Vzhledem k zaměření vyvíjené aplikace je tato dispozice velkou výhodou. Je však třeba zmínit, že WPF 3D engine je původně zamýšlen pouze pro potřebu tvorby interaktivních trojrozměrných uživatelských rozhraní. Jednoduchost a přehlednost jsou tedy hlavním faktorem WPF; výkon však ne. Komplexní 3D aplikace, jako jsou například počítačové hry, WPF jednoznačně nezvládne.

Za pomoci 3D jádra byly implementovány a testovány dvě různé metody konstrukce grafu. Liší se jak vzhledem výsledného grafu, tak samozřejmě nároky na

hardwarové prostředky. Jedna z konstrukcí výrazně naráží na výkonostní nedostatky WPF. Výše zmíněné filtry do jisté míry tento problém dokáží minimalizovat.

Pro přehlednost poskytuje každý graf čtyři různé typy pohledu kamery – jeden perspektivní, volně pohyblivý, a tři ortografické (nárys, bokorys a půdorys) známé z CAD aplikací.

1.1. Systémové požadavky

- Microsoft Windows Vista nebo Windows XP SP 2
- .NET Framework 3.5 SP 1
- Microsoft Visual Studio 2008 SP 1
- Microsoft SQL Server 2005 a vyšší

2. Použité technologie

Windows Presentation Foundation

Windows Presentation Foundation je nový grafický subsystém pro operační systém Windows, který je součástí platformy .NET od verze 3.0. Slovo „nový“ může být poněkud zavádějící, protože první oficiální verze WPF se datuje do roka 2007, kdy vyšla třetí verze platformy .NET. Použil jsem jej však proto, že se jedná o zcela nový přístup k vytváření klientských aplikací pro Windows. Stává se tak přímým konkurentem starším Windows Forms, kterým se do jisté míry WPF podobá. Sám Microsoft v poslední době již Windows Forms dále nerozvíjí a soustředí se výhradně na rozvoj WPF v oblasti klientských aplikací.

Standardní aplikace pro Windows staví na dvou zásadních částech operačního systému Windows: GDI/GDI+ a User32. Pro každého vývojáře se jedná o velmi známé technologie, protože se nad nimi programuje už více než 15 let. Ať už se bude jednat o prastarý Visual Basic 6, MFC aplikace v C++ nebo dokonce o výše zmíněné Windows Forms, stále se programuje nad rozhraním GDI a knihovnou User32. Novější systém jako například Windows Forms bude samozřejmě podstatně výkonější než MFC, protože lépe implementuje obaly (*wrapper*) pro interakci s těmito knihovnami. Nemůže však odstranit technické nedostatky komponenty, která byla navržena před více než patnácti lety.

WPF tento přístup zcela ignoruje. Zatímco dosavadní systémy stavějí na GDI, WPF jako grafické jádro používá *DirectX*. Na první pohled se to může zdát poněkud zvláštní, neboť hlavní využití knihoven DirectX bychom hledali spíše v aplikacích zaměřených výhradně na komplexní grafické konstrukce, jako například počítačové hry nebo modelovací programy. Při hlubším zamyšlení ale dojdeme k závěru, že je to naprosto rozumný krok ze strany Microsoftu. Knihovny DirectX dodají aplikaci, kromě bohatých efektů jako například průhlednost nebo anti-aliasing, hardwarovou akceleraci. To znamená, že DirectX se snaží co nejvíce ulehčit práci procesoru a předat ji procesoru na grafické kartě (*GPU*).

Navíc WPF díky přítomnosti knihoven DirectX implementuje 3D engine založený na *Direct3D*, o kterém bude bližší pojednání dále.

XAML

Další významnou novinkou ve WPF je možnost oddělit grafický návrh aplikace od programové logiky. Toho WPF dosahuje vlastní implementací jazyka XML, jež nese název *eXtensible Application Markup Language* (XAML) a slouží výhradně pro deklarativní popis vzhledu aplikace ve formě stromu značek podobně jako například v HTML. Takový popis je poté uložen do souboru s koncovkou .xaml a při překladu programu je „spojen" s kódem okna. Práci s XAML soubory je možné realizovat v designéru v samotném Visual Studiu při tvorbě programu nebo použít specialní aplikace z balíku *Microsoft Expression Studio*, které, mimo jiné, obsahují *WYSIWYG* (*What-You-See-Is-What-You-Get*) editory podobné například *Adobe Dreamweaver*.

Výhodu snadno nalezneme při práci na větším projektu ve vývojovém týmu, kde designéři vytvářejí vzhled aplikace a programátoři se mohou soustředit na samotnou logiku programu.

Je samozřejmě možné napsat celý program bez využití XAMLu, ochudíme se však o jednoduchost a přehlednost, jakou XAML jednoznačně nabízí. Zároveň existuje celá řada aplikací, která byla vytvořena pouze za pomoci XAMLu; hlavně díky podpoře datových vazeb (*data binding*) přímo v XAMLu. Je však zřejmé, že taková aplikace zvládne jen ty nejzákladnější funkce.

Pravý přínos XAMLu je patrný až při použití stylů, kdy designér může definovat různé předpisy vzhledu jednotlivých komponent a aplikace může poté mezi těmito předpisy (někdy označované jako *stylopisy*) velice jednoduchým způsobem (i za běhu aplikace) přepínat. Tyto techniky jsou známé především v oblasti webového programování.

eXpress Persistent Objects

Nativní knihovna pro práci s databázemi prostředí .NET, nazvaná ADO.NET, obsahuje prostředky pro manipulaci s relačními daty; zejména aplikační rozhraní pro jednotlivé databázové systémy, zvané poskytovatelé dat (*data providers*). Každý poskytovatel je určen pro konkrétní databázový systém a obsahuje třídy pro připojení k danému systému, vytvoření příkazu, který má databázový server vykonat a v neposlední řadě objekt pro čtení získaných dat (`Connection`, `Command` a `DataReader`). Každý poskytovatel odlišuje své třídy od ostatních přidáním názvu databázového systému před

jméno třídy; například jmenný prostor `SqlClient` zahrnuje třídy `SqlConnection`, `SqlCommand` atd.

Samotný postup získání kýžených dat je vcelku jednoduchý. Program se připojí k databázi vytvořením instance třídy `SqlConnection`, které předá připojovací řetězec, a zavoláním její metody `Open`. Poté je nutné vytvořit objekt `SqlCommand` a jeho vlastnosti `CommandText` přiřadit dotaz v jazyce SQL uzavřený v textovém řetězci. V závislosti na očekávaném typu dat zavoláme metodu pro vykonání příkazu. Pokud čekáme jednu skalární hodnotu, voláme metodu `ExecuteScalar`, její výsledek předáme nějaké proměnné a naše práce je u konce. V případě, že však očekáváme celé řádky tabulky databáze, zavoláme nejprve metodu `ExecuteReader`, jejíž výsledek poté cyklicky projdeme řádek po řádku a uložíme jednotlivé výsledky do kolekce. Toto je konvenční postup při práci s relačními daty v objektově orientované aplikaci.

Problémem poskytovatelů dat je fakt, že jako výsledek vracejí relační data, tzn. data s odlišnou strukturou než má cílová objektově orientovaná aplikace. Programátor je tedy nucen výsledky dotazu složitě interpretovat do objektového modelu aplikace. Pro ušetření práce byla tedy potřeba vytvořit prostředky, které by takovouto interpretaci výsledků provedly za programátora automaticky. Za tímto účelem byly navrženy nástroje pro mapování relačního modelu dat na model objektový. Tyto nástroje jsou souhrně označovány pojmem *Object-relational mapping* (Obejktově relační mapovaní, zkratka ORM).

eXpress Persistent Objects (dále XPO) je jedním z mnoha přístupných ORM nástrojů v prostředí .NET, který vyvíjí firma Developer Express. Zjednodušeně řečeno se jedná o jakési rozhraní, které převádí strukturu relačního datového úložiště na objektové a naopak. Každé databázové tabulce je přiřazena třída, jejímiž datovými složkami jsou jednotlivé atributy dané tabulky.

Nedílnou součástí platformy XPO je navíc generátor kódu, který na bázi vytvořené databáze automaticky generuje třídy, které reprezentují jednotlivé tabulky. Tento proces je možné realizovat i v opačném směru – z vytvořených XPO tříd může interpret XPO generovat databázi.

Hlavní výhodou XPO je poskytnutí vyšší míry abstrakce nad daty a jejich implementací. Programátor se už tedy nemusí starat o strukturu dat v databázi, poněvadž veškerá práce s daty se provádí ve vyšší vrstvě. Stejný XPO model je navíc možné použít nad různými databázemi, pokud je struktura databáze totožná (přesněji řečeno obdobná –

vývojář musí dodržet jistá omezení daná vlastnostmi platformy XPO a daného databázového systému). V současnosti podporuje XPO až 20 různých databázových systémů.

Dodatečné knihovny

3D Tools

Knihovna od firmy Microsoft, která rozšiřuje WPF 3D o pár komponent, které v základní verzi WPF chybí. V aplikaci byla použita hlavně třída `Trackball` z této knihovny, která umožňuje uživateli pohybem myši otáčet kamerou, případně při podržení pravého tlačítka myši pohled přiblížit nebo oddálit. S příchodem Microsoft .NET Framework 4.0 bylo očekáváno, že Microsoft tuto knihovnu zařadí jako legitimní součást WPF do platformy .NET. Protože je však knihovna 3D Tools stále ve stádiu vývoje, nestalo se tak.

Druhou užitou komponentou této knihovny je třída `ScreenSpaceLines3D`. Ta není v programu použita přímo; vyžaduje ji však knihovna *Media3D*, která je zmíněna níže.

Media3D

Knihovna Media3D, jejíž autorem je americký spisovatel Charles Petzold, obsahuje různé nástroje, které mají programátorovi zpříjemnit práci s WPF 3D. Základní knihovna 3D enginu není příliš obsáhlá; neobsahuje například žádné nástroje pro generaci základních stereometrických primitiv (kvádr, koule, atd.). Media3D rozšiřuje základní WPF o tyto nástroje a přidává další užitečné třídy, jako například třídu pro generaci souřadných os nebo vytvoření a umístění textu v prostoru, což vyvíjená aplikace vyžaduje pro generaci popisků hodnot v grafu.

KMB Analytics

Tato knihovna obsahuje třídy pro vytváření modelů dat. Její využití je popsáno v kapitole 7.

3. Databáze

Aby bylo možné čerpat nějaká data, je nutné se nejprve podívat na strukturu databáze, s kterou má aplikace spolupracovat. Jak bylo řečeno v úvodu, databáze obsahuje naměřené hodnoty různých veličin z konkrétních měřících přístrojů. Nejprve se tedy podíváme, jak přistupovat k jednotlivým přístrojům.

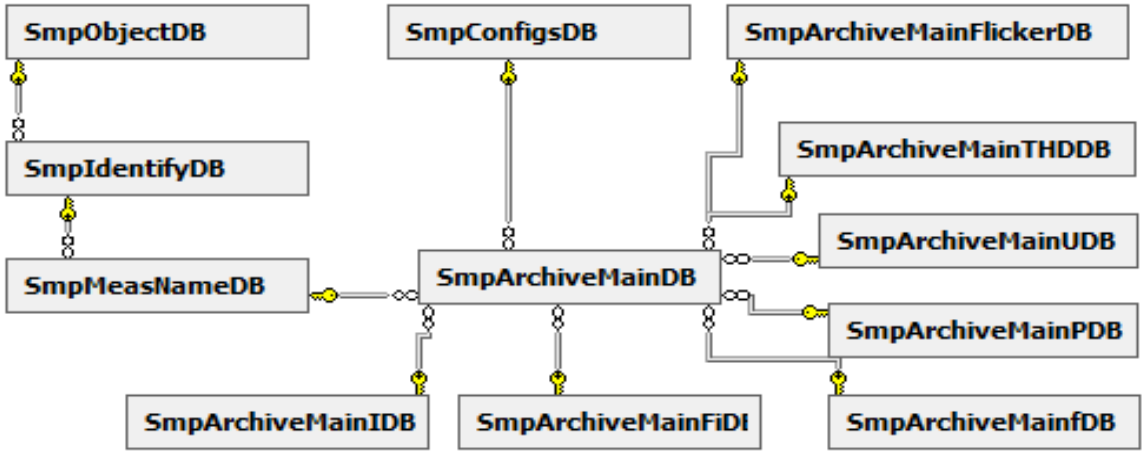
Základním kamenem celé databáze je tabulka s názvem *SmpObjectDB*. Ta obsahuje záznamy o budovách, lépe řečeno o objektech, ve kterých se přístroje nacházejí. Takovým objektem může být právě jedna budova nebo i více budov nějaké firmy. Tabulka obsahuje pouze primární klíč reprezentovaný celočíselnou hodnotou a název objektu. Pod každý objekt spadá kolekce výše zmíněných přístrojů. Ty jsou obsaženy v tabulce *SmpIdentifyDB*, která obsahuje cizí klíč z tabulky objektů a další doplňující informace jako výrobní číslo, verze software atd.

Každému přístroji je pak přiřazeno několik různých měření, která nalezneme v tabulce *SmpMeasNameDB*. Z této tabulky už se dostaneme dále k naměřeným hodnotám.

Nejdůležitější částí databáze je hlavní archiv *SmpArchiveMainDB*. Ten funguje jako jakýsi rozcestník pro příbuzné tabulky, které jsou určeny pro uložení konkrétní veličiny; například tabulka *SmpArchiveMainIDB* ukládá hodnoty proudů, jiná tabulka *SmpArchiveMainUDB* je určena pro uchování informace o naměřených hodnotách napětí atd. Tabulka *SmpArchiveMainDB* tedy obsahuje pouze odkazy do ostatních tabulek. Každý zápis měření probíha po určitém časovém intervalu, proto obsahuje tabulka také čas zápisu (*keyTime*), který je zároveň součástí složeného primárního klíče. Jeho druhou částí je identifikátor měření *keymeasName*.

Zde je třeba podotknout, že jednotlivé řádky v hlavním archivu neodkazují na konkrétní naměřené hodnoty získané v přesném časovém okamžiku měření, ale pouze na minimální, maximální a průměrné hodnoty jednotlivých veličin v časovém intervalu *keyTime* až *endTime*. Atribut *endTime* tedy označuje koncový čas každého takového intervalu. Veškeré naměřené hodnoty jsou ve skutečnosti zkomprimované ve sloupci *Data* typu *varbinary(max)*.

Posledním důležitým atributem tabulky hlavního archivu je cizí klíč směřující do tabulky *SmpConfigsDB*, která obsahuje informace o nastavení. Pro názornost bude lepší si celé schéma ukázat na obrázku:



Obrázek 3-1:Schéma databáze

Databáze obsahuje samozřejmě mnoho dalších tabulek, jejich podrobný popis však není nutný a sahá nad rámec této práce.

4. 3D engine

Dalším logickým krokem při tvorbě programu bylo vybrat vhodný grafický engine. Možností existuje celá řada; postupně projdeme ty nejvýznamnější z nich. Protože aplikace je z principu navržena pouze pro operační systém Windows, vynechávám aplikační rozhraní *OpenGL*.

Direct3D API

Aplikační rozhraní Direct3D je základní součástí knihoven DirectX, které je přístupné pouze pro operační systémy Windows (Windows 95 a výše). Jeho největší výhodou je dozajista nejvyšší výkon ze zde zmíněných knihoven. Takový výkon však s sebou nese vysokou daň a tou je velmi komplexní práce s touto knihovnou. Rozhraní pracuje na nízké úrovni, kde v podstatě jakákoli operace s nějakým objektem vyžaduje maticové transformace. Další nevýhodou je použitý programovací jazyk C++, který efektivitu práce ještě dále snižuje.

Direct3D je dnes zřejmě nejpoužívanějším aplikačním rozhraním v obasti komplexních trojrozměrných aplikací (zejména počítačové hry); jeho použití však shledávám jako zbytečné vzhledem k zaměření aplikace a obtížnosti práce s ním.

Managed DirectX

Managed DirectX (MDX) je zjednodušeně řečeno soubor obalů pro existující rozhraní DirectX určený speciálně pro vývoj na platformě .NET. Výhodou je tedy možnost využití moderních programovacích jazyků C# a VB.NET a knihoven platformy .NET Framework. Tato abstrakce má samozřejmě za následek negativní dopad na výkon tohoto 3D enginu oproti nativním DirectX v C++.

Poslední verze DirectX podporovaná knihovnou MDX je DirectX 9.0, protože firma Microsoft zastavila vývoj MDX, čímž se stává nevhodným kandidátem pro výběr 3D enginu. Jakýmsi oficiálním nástupcem MDX je prostředí *Microsoft XNA*.

Microsoft XNA

XNA (zkratka ve skutečnosti znamená „XNA is Not an Acronym“) je soubor nástrojů a běhové prostředí od firmy Microsoft, které má nahradit knihovny Managed DirectX. Hlavní myšlenkou XNA je poskytnout programátorům zjednodušené vývojové prostředí aplikačního rozhraní DirectX zaměřené výhradně na vývoj videoher. To ovšem neznamená, že by se XNA omezovalo pouze na hry.

Hlavní výhodou XNA je přímá podpora vytvořené aplikace na platformách Xbox 360 a Windows Phone 7. Zároveň XNA disponuje velkou uživatelskou komunitou a velice rozsáhlou uživatelskou podporou.

Jelikož je XNA zaměřeno výhradně na hry, neexistuje žádná přímá podpora XNA obsahu ve formulářové aplikaci (Windows Forms, WPF). Jakýkoli grafický obsah vytvořený v XNA musí tedy běžet ve vlastním dedikovaném okně. Tento neduh lze obejít úpravou XNA okna tak, že se odstraní titulek okna a jeho okraje a následně je takové okno virtuálně ukotveno v klientské oblasti okna WPF aplikace. Celý postup je dosti zdlouhavý a programátor se setká s celou řadou problémů při takovém používání XNA, proto jsem se rozhodl XNA nepoužít.

SlimDX

SlimDX je podobně jako XNA nástupcem Managed DirectX s tím rozdílem, že se jedná o open source projekt od nezávislé firmy. Podobné principy práce s enginy XNA a MDX se tedy dají aplikovat i při vývoji ve SlimDX. K vývoji je možné využít jakýkoli programovací jazyk kompatibilní s platformou .NET včetně jazyků *F#* a *IronPython*.

Zatímco MDX a XNA podporují rozhraní DirectX pouze do verze 9, SlimDX implementuje i knihovny DirectX 10 a DirectX 11. Zároveň je SlimDX jediným rozhraním DirectX na platformě .NET Framework, které plně podporuje .NET 4.0.

SlimDX zároveň podporuje interoperabilitu s WPF. Té je dosaženo WPF elementem `D3Dimage`, který umožňuje vyhradit v okně aplikace prostor, do kterého může program vykreslovat Direct3D obsah. `D3Dimage` funguje samozřejmě i v kombinaci s nativním rozhraním DirectX, nicméně pouze se SlimDX může programátor užít řízený (*managed*) kód. Podpora `D3Dimage` bohužel sahá jen do verze 9 rozhraní DirectX; DirectX 10 a DirectX 11 tedy zůstane v případě použití WPF nevyužito.

Nebýt faktu, že WPF disponuje vlastním vestavěným 3D enginem, volba by s největší pravděpodobností padla na výše zmíněné SlimDX. Protože však při použití 3D enginu WPF není třeba řešit žádné konflikty knihoven, rozhodl jsem se pro WPF.

Nativní engine WPF

Jak bylo řečeno ve druhé kapitole, jádrem grafického subsystému Windows Presentation Foundation je Microsoft DirectX. Není tedy náhodou, že WPF implementuje vlastní aplikační rozhraní pro vykreslování trojrozměrné grafiky. Smyslem tohoto rozhraní není vytváření příliš komplexních grafických aplikací, ale poskytnout programátorovi možnost integrovat 3D grafiku do svých aplikací. Takové použití 3D grafiky může nabýt formy trojrozměrných ovládacích prvků, jednoduchých animací nebo například zobrazení komplexních informací s pomocí třetího rozměru. Zkrátka řečeno, WPF 3D je navrženo jako prostředek k vylepšení uživatelského rozhraní.

S WPF 3D se pracuje podobně jako s jinými aplikačními rozhraními v řízeném kódu. Je zde však několik odlišností; některé zmíním nyní, jiné vyplynou dále v textu. První a nejvíce patrnou odlišností je celková jednoduchost celého rozhraní ve srovnání s ostatními knihovnami. Knihovny WPF jsou navrženy tak, aby programátorovi co nejvíce ulehčily práci s trojrozměrnou grafikou.

Další odlišností je integrace tříd enginu do značkovacího jazyka XAML. Každý objekt knihovny WPF 3D má zároveň svoji reprezentaci v XAMLu, tudíž je možné vytvořit kompletní scény nejen v kódu užitého programovacího jazyka (C#/VB.NET), ale i v XAMLu. Zajímavým využitím této vlastnosti je například vytvoření definice grafického objektu v jazyce XAML, která se následně uloží jako zdroj (*resource*) aplikace. Tento zdroj je poté možné volat v kódu aplikace jako běžný zdroj (například obrázek nebo nějaký řetězec).

V neposlední řadě je třeba doplnit, že WPF 3D je velice úzce svázáno se svým rozhraním pro vykreslování dvourozměrné grafiky. Například třídu pro obyčejnou barvu (přesněji řečeno štětec) `SolidColorBrush` je možné použít jako difúzní materiál 3D objektu.

Poslední zajímavostí je podpora datových vazeb. Je například možné definovat nějaký objekt, tomuto objektu přiřadit transformaci a poté vytvořit datovou vazbu mezi takovou transformací a nějakým ovládacím prvkem; poloha jezdce ovládacího prvku `Slider` může například ovlivňovat rotaci krychle pomocí jednoho řádku kódu.

5. Hlavní okno aplikace

První věc, kterou je nutné po vytvoření nového projektu Windows Presentation Foundation ve Visual Studiu udělat, je definice vzhledu hlavního okna aplikace. K tomuto účelu využijeme značkovacího jazyka XAML, ve kterém se tento proces dá zvládnout mnohonásobně rychleji než v kódu jazyka C#. Ještě rychlejší cestou je použití vestavěného designéra ve Visual Studiu a jednotlivé komponenty postupně přetahovat na okno z nabídky nástrojů. Osobně však tento postup neprosazuji, protože designér automaticky nastavuje všem nově umístěným komponentám pevné rozměry a pozice v okně. Ochudíme se tak o jednu ze silných stránek WPF – *relativní pozicování*.

Další zajímavost, které si po otevření souboru Window1.xaml všimneme, je struktura nápadně podobná jazyku XML nebo HTML. Vzhledem k tomu, že je XAML jen implementací XML, není se čemu divit. S návrhem budeme tedy pracovat v podobě stromové struktury; první prvek tohoto stromu se nazývá kořen nebo kořenový element. Kořenem tohoto souboru je objekt okna – `Window`.

Poznámka: Protože kompletní popis definice vzhledu okna by vydal na desítky stran a obsahově by byl poněkud nudný, zaměřím se pouze na klíčové oblasti. Jednotlivé komponenty budu označovat jako objekty nebo elementy. Oba termíny jsou validní, protože téměř každý element XAMLu je zároveň objektem v jazyce C#/VB.

Před samotným plněním okna by bylo dobré si nejprve nastínit teoretický návrh jeho obsahu. Budeme chtít, aby v horním okraji okna bylo nějaké menu. K dolnímu okraji by bylo vhodné umístit stavový řádek, který bude obsahovat doplňující informace, například pozici kurzoru myši apod. Toto je považováno za jakýsi základ, který by měla mít každá klientská aplikace. Například designér *Java Swing* ve vývojovém prostředí *NetBeans* vytvoří takový návrh hlavního okna aplikace hned při vytvoření nového projektu Java Swing aplikace.

Do hlavní části okna, která vytvoří zbytek klientské oblasti, bude zapotřebí nějaké kreslící plochy pro grafy a vedle této plochy navíc umístit nabídku pro výběr měřícího přístroje a veličiny, jejíž průběh se má vykreslit. K vymezení časového úseku průběhu poslouží kalendář; ten usadíme poblíž nabídky pro výběr přístroje. Návrh tedy vypadá přibližně takto:

Obrázek 5-1:Návrh hlavního okna aplikace

Do elementu `Window`, který představuje okno aplikace, můžeme nyní vkládat další objekty stejně jako v jiných značkovacích jazycích; jsme však omezeni počtem vložených objektů. Do objektu `Window` je totiž možné umístit pouze jeden element. To se může jevit poněkud neprakticky, ale použitím panelů, jejichž funkci vzápětí vysvětlím, se tento problém eliminuje. Důvodem minimalizace povoleného počtu vnořených prvků na jeden je vlastnost `Content` objektu `Window`; té je možné přiřadit pouze jeden objekt libovolného datového typu, protože se jedná o obyčejnou proměnnou typu `object`.

Panely (Layout Panels) jsou obecně speciální komponenty plnící funkci kontejneru, do kterého se vkládají další elementy. Pokud tedy do elementu `Window` vnoříme nějaký panel, obejdeme omezení vlastnosti `Content`. Jaký panel ale nyní zvolit? WPF jich nabízí celou řadu a každý disponuje specifickou funkcí; shrňme si je do tabulky:

`Canvas`	Umisťuje elementy dle souřadnic (vykreslování)
`DockPanel`	Funkce ukotvení ke kraji (Top, Left, Right, Bottom)
`StackPanel`	Skládá elementy pod sebe nebo vedle sebe
`WrapPanel`	Podobný StackPanelu, ale zalamuje konce řádku/sloupce
`Grid`	Flexibilní mřížka s určitým počtem řádků a sloupců
`UniformGrid`	Podobný Gridu, ale buňky mají stejné rozměry

Tabulka 5-1: Panely rozvržení

Protože potřebujeme ke krajům okna ukotvit menu a stavový řádek, použijeme jako výchozí panel `DockPanel`. Umístíme tedy `DockPanel` do elementu `Window`. Menu (element `Menu`) vložíme do panelu `DockPanel` a nastavíme ukotvení na `DockPanel.Dock="Top"`. Analogicky vytvoříme objekt `StatusBar` pro reprezentaci stavového řádku a nastavíme mu ukotvení ke spodnímu okraji.

Posledním elementem v panelu `DockPanel` bude celá hlavní část okna. Protože tato oblast čítá celkem čtyři další prvky, využijeme opět panelu. Pro tento účel nejlépe poslouží mřížka – `Grid`. Té nastavíme ukotvení k levému okraji obrazovky. Protože panel `Grid` musí znát počet řádků a sloupců, které bude obsahovat, je třeba tyto vlastnosti definovat. Definice řádků a sloupců se v jazyce XAML uvádí jako vnořené elementy v mřížce `Grid.RowDefinitions`, resp. `Grid.ColumnDefinitions`. Do těchto elementů se poté vkládají jednotlivé definice. Ukažme si krátký příklad:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Width = "50" Height = "Auto"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button  Grid.Row = "0" Grid.Column = "1"/>
</Grid>
```

Zdrojový kód 5-1: Definice panelu Grid

Zde jsme explicitně definovali pouze jeden řádek mřížky. Pokud nejsou v panelu `Grid` přítomny žádné definice, nastaví se automaticky počet řádků a sloupců na jeden. Každému prvku uvedenému po definicích je poté nutné určit v jaké buňce mřížky se nachází – `Grid.Row = "0" Grid.Column = "1"`. Indexy jsou číslovány od nuly a pokud explicitně neuvedeme pozici v mřížce, nastaví se sama do prvního sloupečku prvního řádku.

`Grid` bude sestávat z jednoho řádku a dvou sloupců. V první části (vlevo) budou umístěny nabídky pro výběr přístroje a veličiny a kalendář. Pravou část mřížky rezervujeme ploše, na které poté budeme vykreslovat průběh měření. Protože do levé části

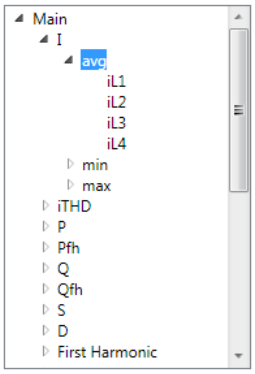
chceme dosadit celkem tři prvky, vytvoříme v prvním sloupci mřížky další panel `Grid` o jednom sloupci a třech řádcích. Do každého řádku vložíme postupně objekt `Calendar` a dva elementy `TreeView`; jeden pro nabídku přístrojů, druhý k výběru veličiny. Do pravého sloupce nadřazeného panelu `Grid` nakonec umístíme poslední `Grid`.

Tento `Grid` je třeba rozdělit na čtyři stejně veliké podoblasti – dva řádky, dva sloupce. Do každé takto vzniklé buňky vložíme element `Viewport3D`. Objekt `Viewport3D` vytvoří v okně virtuální 3D prostor, do kterého je možné vkládat trojrozměrný obsah. Právě do těchto objektů bude následně aplikace vkládat grafy (celkem tedy čtyři, respektive jeden graf zobrazený z různých úhlů pohledu kamery).

Protože může nastat situace, kdy se veškerý obsah nevejde do omezené klientské oblasti okna (uživatel například zmenší okno programu), budeme chtít do `Gridu` nějakým způsobem implementovat posuvníky. K tomuto účelu existuje ve WPF komponenta `ScrollViewer`, do které je možné vnořit libovolný element WPF. Tomuto vnořenému elementu pak `ScrollViewer` dodá potřebné posuvníky, pokud element přesahuje velikost `ScrollVieweru`. `Grid` tedy vložíme do objektu `ScrollViewer` a tomu nastavíme pevné rozměry. Pokud tak neučiníme, nebude `ScrollViewer` fungovat správně. Tím máme celý design okna prakticky hotový.

Vraťme se však ještě k objektům `TreeView`. Komponenta `TreeView` zobrazuje hierarchická data ve stromové struktuře. Vzhledem ke struktuře databáze se nám tyto elementy budou hodit. Aplikace čítá celkem dva. První poslouží k zobrazení přístrojů. Jelikož tento `TreeView` musíme zaplňovat dynamicky v okamžiku připojení k databázi, což XAML nezvládne, necháme tento element prozatím prázdný.

Druhý element `TreeView` můžeme naplnit už předem v kódu XAML, protože měřené veličiny jsou pro všechny přístroje databáze totožné. Jelikož tento `TreeView` čítá desítky položek, uvedeme si pro ilustraci pouze základní strukturu. Začněme obrázkem:



Obrázek 5-2:Struktura objektu TreeView

V kořeni stromu se nachází název archivu. Po výběru archivu se rozbalí nabídka s jednotlivými veličinami. Na dalších úrovních už jen dále upřesňujeme výběr. Například na obrázku máme rozbalenou nabídku s elektrickým proudem; ta obsahuje položky pro průměrné, minimální a maximální naměřené proudy. Každá tato položka dále zobrazí čtveřici měřených proudů. Analogicky se zobrazují i ostatní veličiny.

Tato hierarchie se v jazyce XAML zapíše postupným vnořením elementů `TreeViewItem` do `TreeView`; postup si demonstrujeme na ukázce.

```
<TreeView>
   <TreeViewItem Header="Main">
      <TreeViewItem Header="I">
         <TreeViewItem Header="avg">
            <TreeViewItem Header="iL1"/>
            <TreeViewItem Header="iL2"/>
            <TreeViewItem Header="iL3"/>
            <TreeViewItem Header="iL4"/>
         </TreeViewItem>
         <TreeViewItem Header="min">
            <TreeViewItem Header="iL1"/>
            <TreeViewItem Header="iL2"/>
      ...
```

Zdrojový kód 5-2: Struktura komponenty TreeView

Tímto máme celý vzhled hotový a můžeme se zabývat kontrukcí vlastních grafů.

6. 3D Grafy

Data v databázi ENVIS jsou závislá pouze na čase. Bylo tedy nutné vymyslet způsob, jakým se budou tato data dimenzovat, aby vytvořila trojrozměrný objekt. V praxi bývá zvykem například při použití barevných map (*spektrogram*) rozdělit časová data na kratší úseky stejné nebo podobné délky a tyto úseky vyskládat po řádcích do mapy. Tímto způsobem získají data potřebnou druhou nezávislou dimenzi. Třetí rozměr se poté simuluje využitím barvy tak, že jednotlivé úrovně hodnot se zobrazí do předem dané škály barev.

Z tohoto principu bude vycházet konstrukce 3D grafu vyvíjené aplikace. Za předpokladu, že úseky dat mají být stejně dlouhé, byla stanovena délka jednotlivých úseků dat na jeden den. Délka úseků v jiných aplikacích často bývá proměnná, existují například vizualizace dat, kde jako délka úseku je zvolen 1 týden nebo 1 měsíc. Tato proměnlivost však přináší při konstrukci 3D grafu jisté komplikace; proto pro jednoduchost byla zvolena délka jednoho dne jako základní jednotka jedné dimenze.

Princip konstrukce 3D objektů ve WPF

Aby bylo možné tvořit 3D grafy, je nejprve nutné vysvětlit, jak vůbec funguje aplikační rozhraní WPF při vytváření trojrozměrných útvarů. Zároveň bude v této kapitole vysvětlen princip souřadného systému WPF a element `Viewport3D`.

Viewport3D

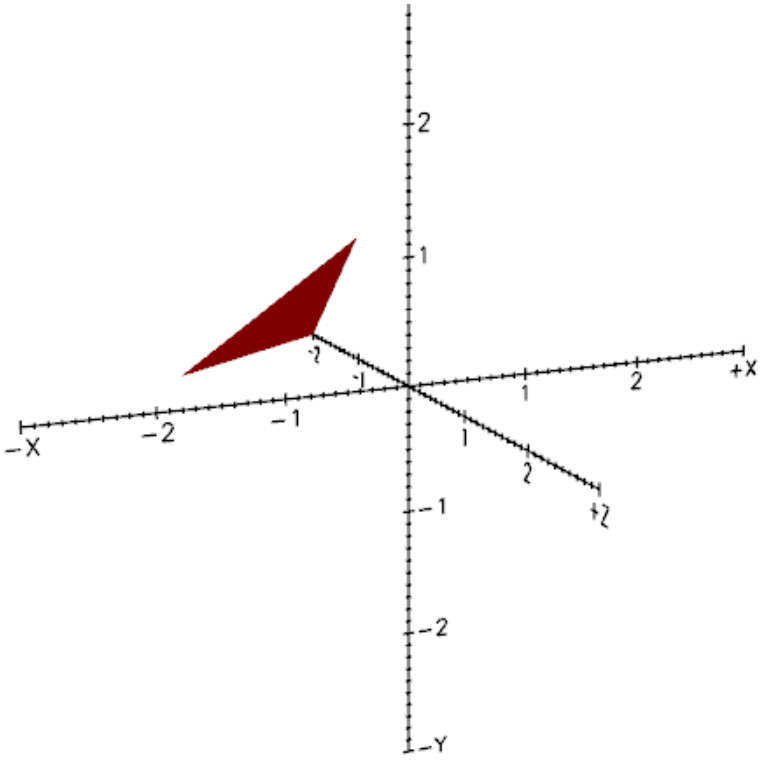
`Viewport3D` je objekt, který dokáže uchovávat a zobrazovat trojrozměrné útvary vytvořené ve WPF 3D. Vytvoří jakýsi virtuální prostor, který může uživatel sledovat kamerou, kterou definuje `Viewport3D`. Taková kamera funguje v podstatě jako snímač běžné kamery. Bližší informace o kamerách jsou v podkapitole Kamera.

Objekt `Viewport3D` se chová jako kterýkoli jiný ovládací prvek WPF, tudíž jej stačí umístit na formulář a už tehdy je připraven k zobrazení 3D obsahu.

Poznámka: Objekt Viewport3D označuji v textu pro jednoduchost jen jako „Viewport“ a zároveň jej na mnohých místech skloňuji podle pravidel českého jazyka podle vzoru „hrad“.

Souřadnicový systém WPF

WPF používá klasický systém tří souřadných os *X*, *Y* a *Z*, které se protínají v počátku. Malou odlišností od rozhraní DirectX je použitá konvence značení. Osa *Y* totiž odpovídá výšce, nikoli hloubce prostoru. Tento způsob je použit kvůli úzké souvislosti mezi trojrozměrnou a dvourozměrnou grafikou ve WPF. Zároveň je zvykem stavět kameru před plochu *XY*, tzn. do kladných hodnot na ose *Z*, a do záporných hodnot osy *Z* (za plochu *XY*) klást sledované objekty. Pro přehlednost zde uvedeného následuje obrázek:



Obrázek 6-1:Souřadnicový systém WPF

Konstrukce objektů

Každý objekt v 3D prostoru lze vymodelovat souborem trojúhelníků. Trojúhelník je nejjednodušší zobrazitelná plocha v prostoru, proto funguje jako základní jednotka každého útvaru.

Pro vytvoření geometrie objektu existuje ve WPF objekt `MeshGeometry3D`, který definuje (mimo jiné) dvě velice důležité vlastnosti: `Positions` a `TriangleIndices`. `Positions` je kolekce objektů typu `Point3D`; do této kolekce ukládáme body (*vertexy*), ze kterých se následně budou tvořit jednotlivé trojúhelníky objektu. `Point3D` je objekt reprezentující bod v trojrozměrném prostoru.

Druhá významná vlastnost, `TriangleIndices`, je kolekce typu `Int32Collection` a obsahuje tři celá čísla pro každý trojúhelník obsažený v útvaru. Tato čísla jsou indexy bodů v kolekci `Positions`. Ukažme si definici objektu, který sestává z jediného trojúhelníku:

```
<MeshGeometry3D Positions="-1.5 0 -1, 0 1 -1, 0 0 -2"
                TriangleIndices="0 1 2" />
```

Zdrojový kód 6-1:Ukázka definice objektu MeshGeometry3D

Zde je třeba dávat pozor na pořadí, v jakém zadáváme hodnoty do kolekce `TriangleIndices`. Každý trojúhelník má totiž dvě strany: přední a zadní. Právě pořadí indexů v kolekci `TriangleIndices` definuje, která strana trojúhelníku bude stranou přední. WPF používá pro určení přední strany „*pravidlo pravé ruky*“. Pokud tedy hypoteticky přiložíme pravou ruku k tvořenému trojúhelníku a pokrčíme prsty, vztyčený palec ukazuje směr vektoru vycházející z přední strany trojúhelníku. Jedná se tedy o *normálový vektor* trojúhelníku. Důvodem této vlastnosti je vykreslování. Bývá totiž dobrým zvykem vykreslovat pouze přední stranu; zejména v uzavřených objektech, kde zadní strana nebude viditelná, je toto výhodná praktika, poněvadž ušetříme prostředky aplikace.

Geometrie grafů

Pro vytvoření trojrozměrných vizualizací naměřených dat bylo třeba navrhnout různé metody konstrukce grafů. Celkem byly vytvořeny dvě různé metody konstrukce; liší se od sebe vzhledem i výpočetní náročností a nároky na paměť operačního systému. V následujícím textu se s nimi seznámíme blíže.

Sloupcová geometrie

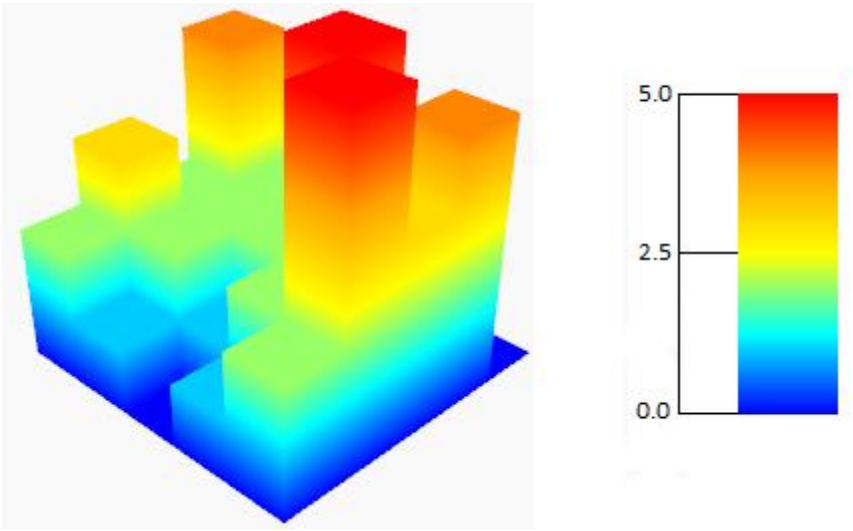
Tato konstrukce je velice podobná sloupcovému grafu; jedné hodnotě tedy odpovídá jeden sloupec. Každý sloupec je zastoupen kvádrem proměnné výšky závislé na naměřené hodnotě. Jednotlivé bloky však leží bezprostředně vedle sebe, nemají tedy mezi sebou žádný volný prostor na rozdíl od klasického sloupcového grafu.

Nyní tedy k samotnému postupu, jak aplikace takový graf vytvoří. Vyvíjená aplikace obsahuje statickou třídu s názvem `Paint3D`, ve které se nachází obě metody pro vytvoření konstrukce grafu včetně metody `CreateBlockGeometry`. Ta postupně

prochází vstupní data, vyhodnocuje rozměry kvádrů v závislosti na naměřené hodnotě a volá metodu `CreateBlock`, která se postará o vytvoření daného kvádru. Tato metoda se nachází ve statické třídě `Utility`.

Metoda `CreateBlockGeometry` při každém kroku činnosti zároveň kontroluje, zda nedošla na konec dne a v takovém případě začne další kvádry skládat o patřičnou šířku podstavy kvádru dál ve směru osy *Z* (hloubka). Tímto způsobem metoda dojde na konec dat a vytvořený objekt vrátí jako objekt typu `Model3DGroup`, což je soubor jednotlivých kvádrů zabalený do jednoho objektu. Výhodou objektu `Model3DGroup` je společný hraniční rámec (*bounding box*) pro všechny kvádry, tudíž se s ním dá pracovat jako s jedním objektem. Tato vlastnost přijde vhod při vytváření souřadných os a popisků grafu.

Dále je důležité kvádry nějakým způsobem obarvit, aby na první pohled vypovídaly o hodnotě, kterou reprezentují. Ideální je využít nějaké barevné palety a tu podle hodnot mapovat na jednotlivé kvádry. Dá se říci, že standardně se používá barevná škála *modrá-azurová-zelená-žlutá-oranžová-červená*. Pro ilustraci uvedeme obrázek:

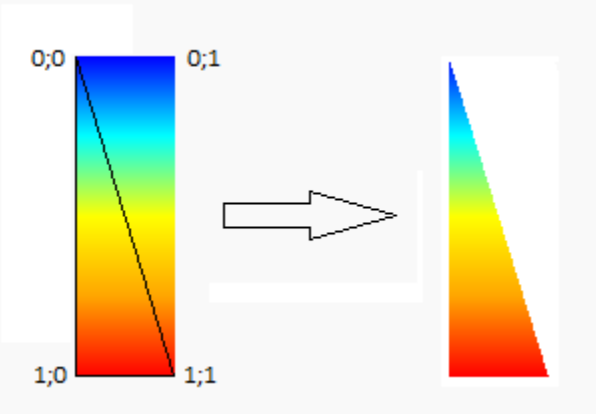


Obrázek 6-2:Mapování barevného přechodu

Na obrázku je vidět, jakým způsobem má probíhat mapování barevného gradientu. Modrá barva značí nejnižší hodnoty, červená naproti tomu ty nejvyšší. K takovému mapování využijeme třídu WPF zvanou `LinearGradientBrush`, která slouží k vytvoření plynulého přechodu barev. Výsledná barva se poté dá použít jako textura, kterou namapujeme na každý kvádr; vždy podle velikosti kvádru určitou část textury.

Protože každý trojrozměrný objekt je ve WPF (a v jakémkoli jiném 3D enginu) tvořen trojúhelníky, musí být každý takový trojúhelník objektu samostatně namapován na

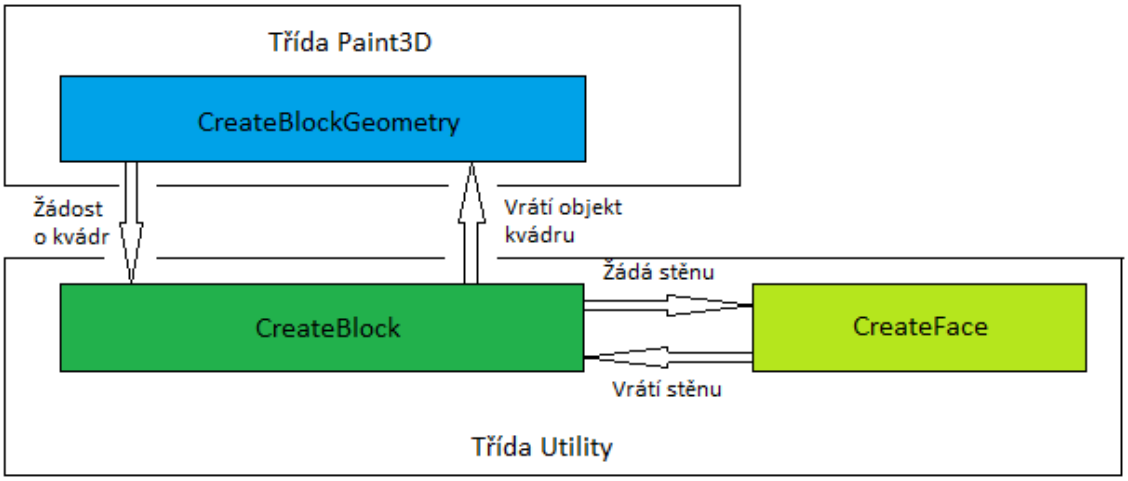
texturu. Pro tento úkon existuje ve třídě `MeshGeometry3D` vlastnost `TextureCoordinates`, což je kolekce bodů dvourozměrného prostoru. Do této kolekce postupně ukládáme relativní souřadnice bodů trojúhelníků tak, jak se mají namapovat na danou texturu. Hodnoty souřadnic se pohybují v rozmezí *(0; 0)* až *(1; 1)*, kde bod *(0; 0)* označuje horní levý roh textury a bod *(1; 1)* se nachází v pravém dolním rohu. Příklad mapování jednoho trojúhelníku je uveden zde na obrázku:



Obrázek 6-3:Mapování jednoho trojúhelníku na texturu

V souvislosti s mapováním gradientu je třeba se zmínit o konstrukci kvádru. Principielně se samozřejmě jedná o banální záležitost. Kvádr je složen ze šesti stěn, každá stěna se skládá ze dvou trojúhelníků; teoreticky tedy stačí náležitě spojit 8 vertexů trojúhelníky tak, že každý bod je společný třem stěnám kvádru. Problém však nastává v okamžiku samotného mapování, protože neexistuje topologie rozložení trojúhelníků na textuře tak, aby byly obarveny všechny stěny kvádru.

Bylo tedy nutné každou stěnu reprezentovat jako samostatný objekt a každý mapovat na vlastní texturu. Pro tento úkon je ve třídě `Utility` přítomna statická metoda `CreateFace`, která přijímá 4 souřadnicové body stěny jako parametry. Tuto metodu volá funkce `CreateBlock` celkem šestkrát – jednou pro každou stěnu. Konečný systém volání metod pro vytvoření sloupcového grafu tedy vypadá takto:



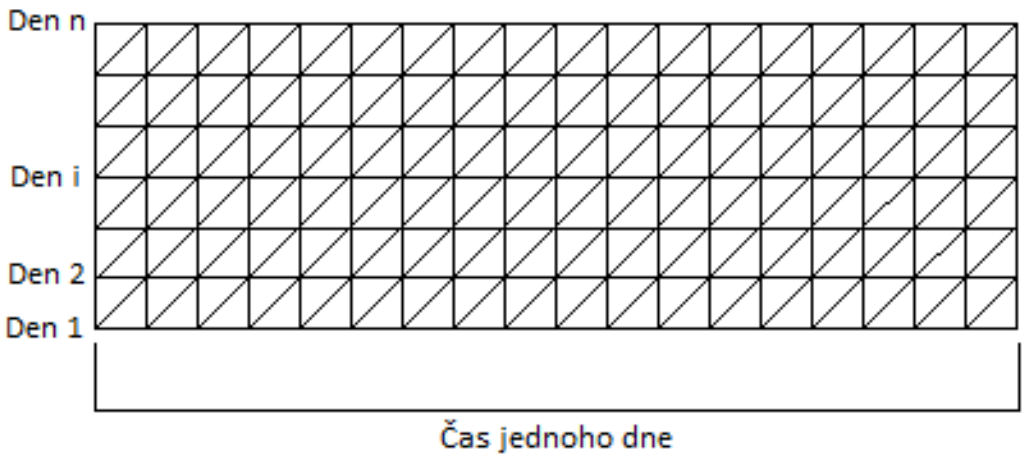
Obrázek 6-4:Systém volání metod konstrukce sloupcového grafu

Výsledná geometrie vypadá přirozeně. Tato metoda ovšem naráží na výkonostní nedostatky 3D enginu WPF. Tento problém řeší další metoda geometrie.

Plošná geometrie

Cílem této metody je vykreslení čitelného grafu s využitím minimálního množství prostředků systému. Konstrukce je tedy velice jednoduchá. Principielně se jedná o velice jednoduchou metodu *triangulace*, kdy každý vertex reprezentuje jednu naměřenou hodnotu. Vertexy jsou vzájemně propojeny tak, aby vytvořily plochu. Souřadnice Y každého vertexu bude záviset na hodnotě získané z databáze.

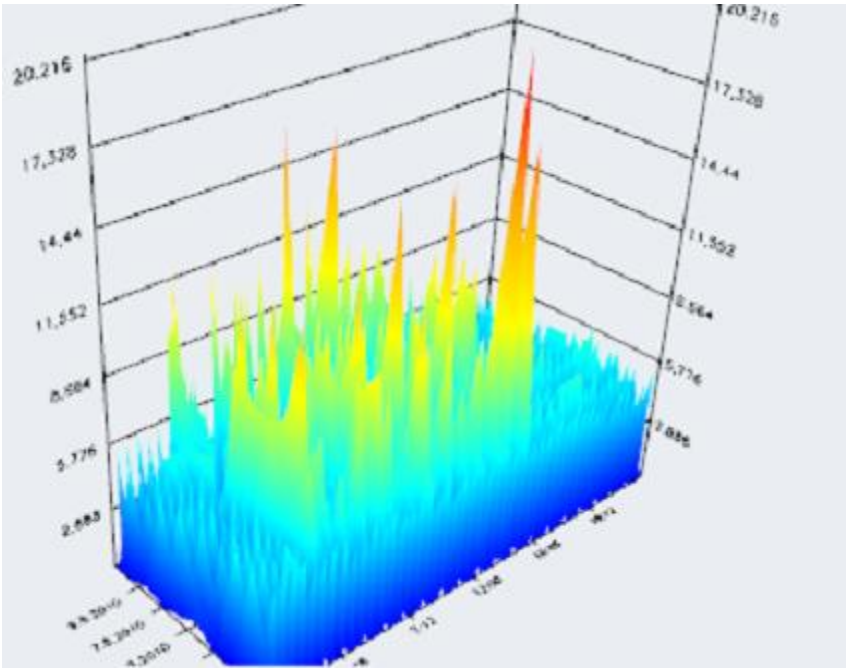
Zjednodušené schéma při pohledu shora na objekt následuje na obrázku:



Obrázek 6-5:Princip topologie plošného grafu

Pro vytvoření plošné geometrie existuje ve třídě `Paint3D` statická metoda `CreateSurfaceGeometry`. Ta má obdobnou funkci jako metoda `CreateBlockGeometry` s tím rozdílem, že nemusí volat žádné další pomocné funkce, protože veškerá konstrukce proběhne v těle této metody. Funkce postupně prochází data a podle času řadí jednotlivé vertexy do objektu a náležitě je spojuje do trojúhelníků dle principu zobrazeného na obrázku 6-5.

Barvení grafu bylo také jednodušší než v případě metody sloupců; celý objekt je namapován na jednu texturu typu `LinearGradientBrush`, kdy algoritmus pouze sleduje výšku vertexu relativně k naměřenému maximu hodnot. Výslednou geometrii vidíme na obrázku 6-6.



Obrázek 6-6:Plošná geometrie grafu

Z obrázku je patrné, že výsledný útvar nepůsobí příliš dojmem aproximované plochy. Na tomto obrázku je totiž ukázka naměřených hodnot odebíraného elektrického proudu, který má bohužel velice proměnlivou charakteristiku v krátkých časových intervalech. Tuto vlastnost eliminuje použití filtrů dat, kterým věnuji samostatnou kapitolu.

Kamera

Kamery jsou speciální třídy, které slouží k nastavení pohledu na sledovanou scénu uvnitř Viewportu. Při sledování trojrozměrného prostoru funguje kamera jako nástroj,

který promítá trojrozměrný svět (3D scéna) na dvourozměrnou plochu, tedy na vykreslovací plochu komponenty `Viewport3D`.

WPF nabízí celkem tři druhy kamer:

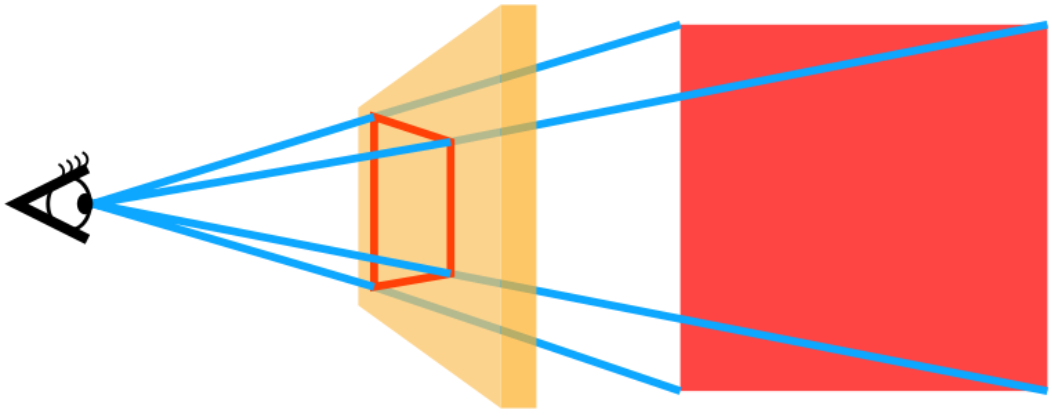
- Perspektivní
- Ortografická
- Maticová

Každý typ kamery je reprezentován vlastní třídou; například třída pro perspektivní kameru se nazývá `PerspectiveCamera`. Třídy `PerspectiveCamera` a `OrthographicCamera` dědí od abstraktní třídy `ProjectionCamera`, která definuje dvě důležité vlastnosti: `Position` a `LookDirection`.

Vlastnost `Position` je typu `Point3D` a definuje, kde se kamera nachází v prostoru. Druhá vlastnost, `LookDirection`, je vektorem (typ `Vector3D`), jehož směr udává směr pohledu kamery. Tyto dvě vlastnosti tvoří základní nastavení společné pro ortografickou a perspektivní kameru. Třída `MatrixCamera` těmito vlastnostmi nedisponuje, protože nabízí rozhraní pro návrh vlastní projekce.

Perspektivní projekce

Třída `PerspectiveCamera` realizuje funkci perspektivní projekce. Tento druh projekce je velice podobný principu funkce lidského oka nebo konvenční videokamery. Imaginární světelné paprsky jednotlivých bodů snímaného objektu se sbíhají do společného bodu (někdy zvaný *úběžník*), který se nachází před projekční plochou. V místech, kde se paprsky protínají s plochou jsou vytvořeny body promítnutého objektu.



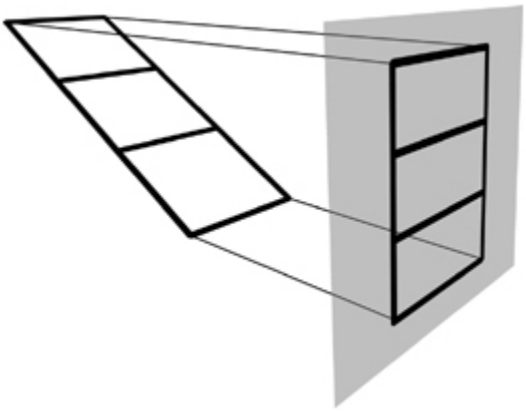
Obrázek 6-7:Perspektivní projekce [7]

Perspektivní kamera simuluje perspektivní deformaci; objekty dále od kamery se jeví jako menší než objekty umístěné blíž ke kameře.

Tento druh projekce byl použit v aplikaci pro perspektivní pohled na grafy. Umístění a směrový vektor pohledu kamery nastaví funkce `SetCameraPerspective` podle tělesových úhlopříček hraničního rámce snímaného objektu. Kamerou lze navíc otáčet a lze ji přibližovat nebo oddalovat použitím myši. Této funkce je dosaženo komponentou z knihovny 3D Tools zvanou `Trackball`.

Ortografická projekce

Chování ortografické (nebo také *ortogonální*) projekce ve WPF realizuje třída `OrtographicCamera`. Princip použití je podobný třídě `PerspectiveCamera`. Kamera ale zachovává velikosti snímaných objektů, a tak i při oddalování objektu od kamery bude tento objekt stále stejně velký.



Obrázek 6-8:Ortografická projekce [2]

Třída `OrtographicCamera` implementuje vlastnost zvanou `Width`, která definuje šířku promítací plochy ortografické kamery. Tato promítací plocha je kolmá na směrový vektor kamery a její střed leží v bodě umístění kamery (`Position`). Šířka této plochy ovlivní snímaný rozsah scény v poměru 1:1 vzhledem k souřadnému systému WPF.

Ortografická projekce má jednu významnou výhodu pro zaměření vyvíjené aplikace. Protože při této projekci nedochází k perspektivnímu zkreslení dimenzí objektů, je velice vhodná pro specifické pohledy na data, u kterých je kritická přesnost. Takové projekce jsou implementovány celkem tři:

- Nárys
- Bokorys
- Půdorys

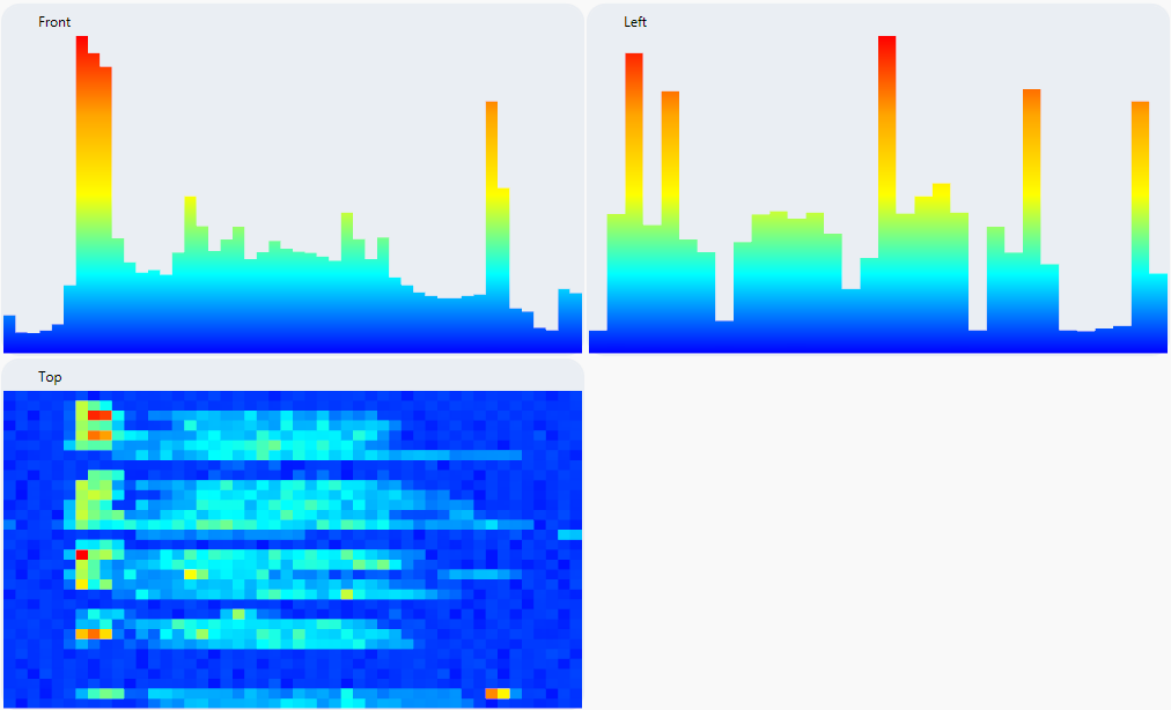
Požadavkem těchto projekcí je zároveň zarovnání okrajů grafu s okrajem projekční plochy; přesněji řečeno s okraji komponenty `Viewport3D`. Jednoduchým způsobem je poté možné k okrajům grafu přidat popisky os. Tohoto efektu lze dosáhnout stanovením vlastnosti `Width` dané ortografické kamery na šířku grafu. Nyní zbývá ještě nastavit výšku projekce. Zde však narážíme na kritický nedostatek objektu `OrtographicCamera` – absence vlastnosti pro definici výšky projekční plochy.

Je tedy nutné zvolit jiný přístup – definovat vlastní projekci tak, aby imitovala projekci ortografickou. K tomuto účelu byla použita maticová kamera. Třída `MatrixCamera` požaduje definici projekční a zobrazovací transformační matice. Tento úkon obstarají dvě statické metody ve třídě `Utility`.

Metoda `SetViewMatrix` přijímá jako parametry požadovanou pozici kamery, směrový vektor pohledu a normálový vektor pohledu kamery. Na bázi těchto parametrů vyhodnotí metoda žádanou zobrazovací matici a předá ji objektu `MatrixCamera`.

Druhá metoda, `SetOrthographicOffCenter`, již vytváří vlastní ortografickou projekci. Výhodou této metody je možnost definovat i výšku projekční plochy na rozdíl od standardní ortografické kamery obsažené v knihovně WPF. Výslednou matici projekce poté stačí předat kameře.

Na následujícím obrázku je ukázka všech tří ortografických projekcí. Grafy zatím neobsahují popisky hodnot; těmi se zabývá následující podkapitola.



Obrázek 6-9:Ortografické projekce grafu - nárys, bokorys, půdorys

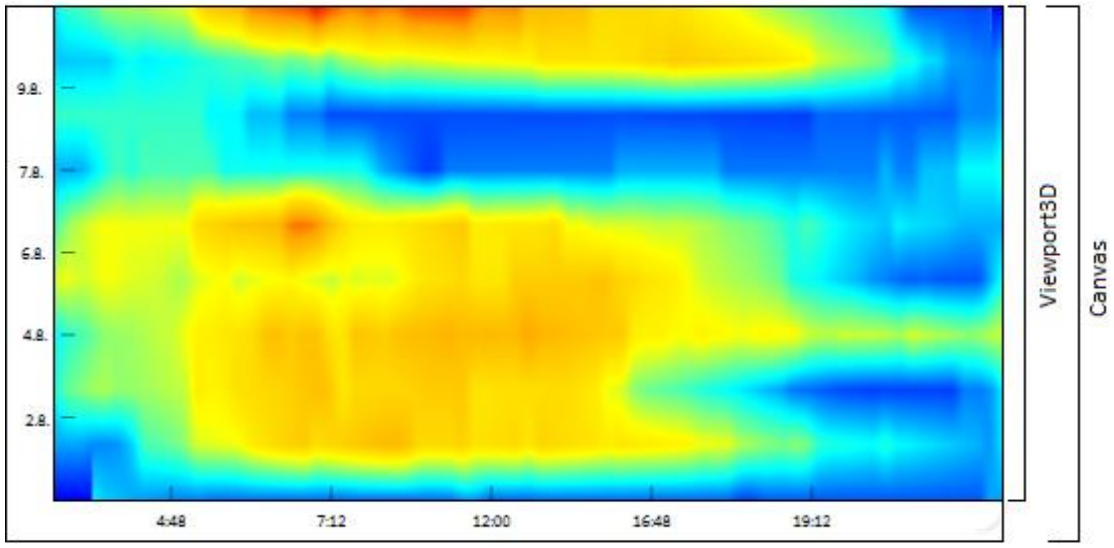
Souřadné osy a popisky grafů

Aby měl graf nějakou informační hodnotu, musí být ještě doplněn o popis zobrazených hodnot. S tím souvisí vykreslení souřadného systému a usazení grafu do jeho počátku.

Jak bylo zmíněno výše, aplikace využívá k vizualizaci čtyř různých pohledů na graf – tří ortografické a jeden perspektivní. Protože ortografické pohledy jsou škálované tak, aby navazovaly přesně na okraje Viewportu, je možné kompletní popis těchto grafů realizovat 2D grafikou. K okrajům Viewportů určených pro ortografické pohledy stačí přidat nějaký vykreslovací prostor a do něj vkládat popisky.

Tento prostor je vytvořen vložením elementu `Canvas` pod `Viewport3D` (ve smyslu hloubky okna – *Z index*). Panel `Viewport3D` je poté nutné náležitě zmenšit, aby okraje panelu `Canvas` byly viditelné a dostatečně velké na vložení textu. Protože popisky se mají vykreslovat u levého a spodního okraje grafu, je `Viewport3D` zarovnán k pravému hornímu okraji panelu `Canvas`.

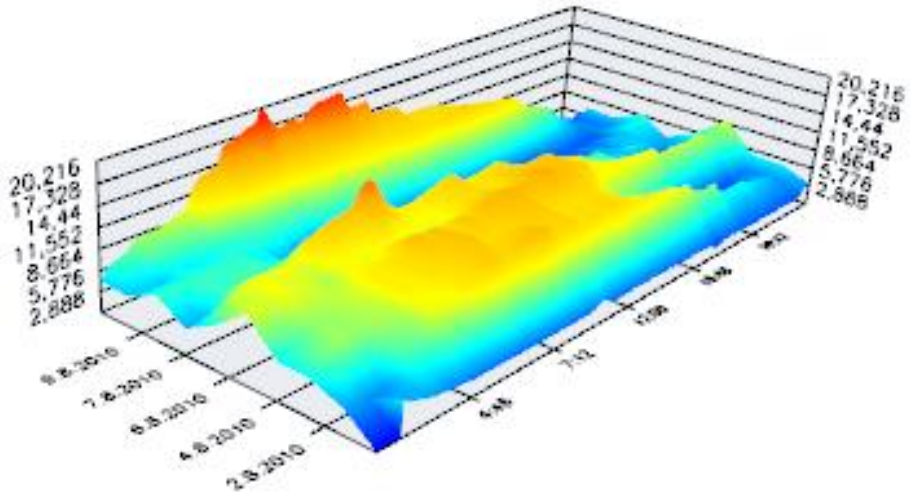
Aplikace poté dle vstupních dat zaplní panel `Canvas` popisky hodnot. Na doprovodném obrázku je zobrazena ortografická projekce pohledu shora na vytvořený graf včetně zobrazených popisků hodnot.



Obrázek 6-10:Ortografická projekce - půdorys grafu s popisky

U perspektivní projekce už tento způsob zobrazení popisků nelze použít, protože aplikace dovoluje uživateli manipulovat s kamerou. Je tedy nutné sestavit souřadné osy a popisky hodnot grafu přímo v trojrozměrném prostoru uvnitř Viewportu. Zde byla využita knihovna Media3D, která obsahuje třídy pro vykreslení přímek a textu do 3D prostoru. Knihovna WPF těmito nástroji bohužel nedisponuje.

Třída `WireLine` knihovny Media3D vytvoří podle zadaných bodů přímku v prostoru. Právě kombinací instancí této třídy je vytvořen souřadný systém. Popisky hodnot grafu poté realizuje třída `WireText`, která umožňuje zobrazit text v 3D prostoru. Celý proces vytvoření souřadného systému a popisků hodnot grafu probíhá v metodě `CreateLabels3D` ve třídě `Utility`. Tato metoda přijímá vstupní data a rozměry už vytvořeného grafu jako parametry a z těchto parametrů vypočítá pozice všech přímek a textů. Výsledný souřadný systém vypadá takto:



Obrázek 6-11:Souřadný systém ve 3D

Tímto končí kapitola o vytváření trojrozměrných grafů ve Windows Presentation Foundation.

7. Modelování dat

Součástí zadání této práce je implementovat různé metody zpracování a filtrace vstupních dat, které se aplikují před samotnou vizualizací. V rámci práce byly vybrány celkem čtyři takové metody – tři statistické a jedna interpolační. Tato kapitola se zabývá jejich popisem.

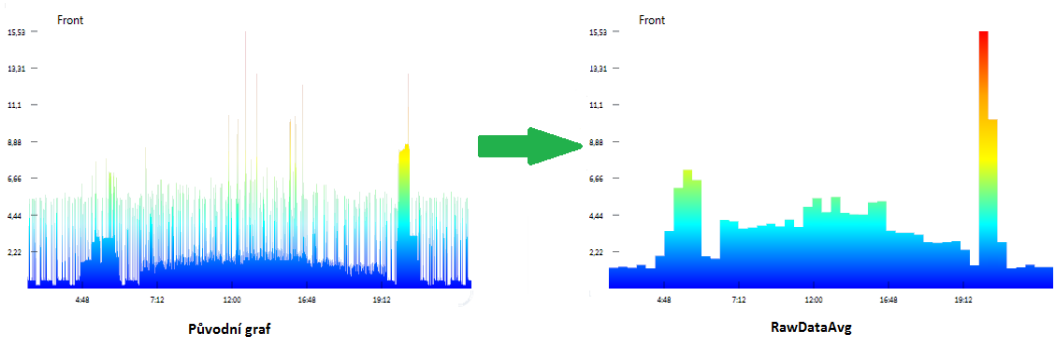
Statistické modely dat

Aplikace pracuje s rozsáhlou databází, která obsahuje velké množství dat. Pro potřebu samotné vizualizace není však vždy potřeba veškerých naměřených hodnot, ale jen nějaké statistické vyhodnocení za delší časový interval. Měřící přístroj může například měřit odběr elektrického proudu jednou za vteřinu. Tímto způsobem bude mít aplikace k dispozici přes 86 tisíc hodnot na jeden den, které k vizualizaci rozhodně nevyužije vzhledem k faktu, že jedno okénko s grafem zabere na obrazovce cca 450 obrazových bodů na šířku a 300 bodů na výšku.

Program tedy implementuje tři statistické modely dat:

- RawDataAvg model
- RawDataMin model
- RawDataMax model

Všechny tři modely pracují na stejném principu. Vstupní data rozdělí na úseky dané délky (například 1 hodina); každý úsek poté vyhodnotí podle své funkce a vrátí jednu hodnotu. Model RawDataAvg například vrací aritmetický průměr hodnot v daném časovém intervalu. Model RawDataMin vrací statistická minima intervalů a RawDataMax maxima těchto intervalů.



Obrázek 7-1:Použití statistického modelu RawDataAvg

Tímto způsobem dojde k redukci dat a zjednodušení celé vizualizace. Jako vedlejší efekt tyto modely navíc snížují nároky na systémové požadavky, což je velice výhodné, zejména při použití sloupcové geometrie (viz kapitola 6) grafu.

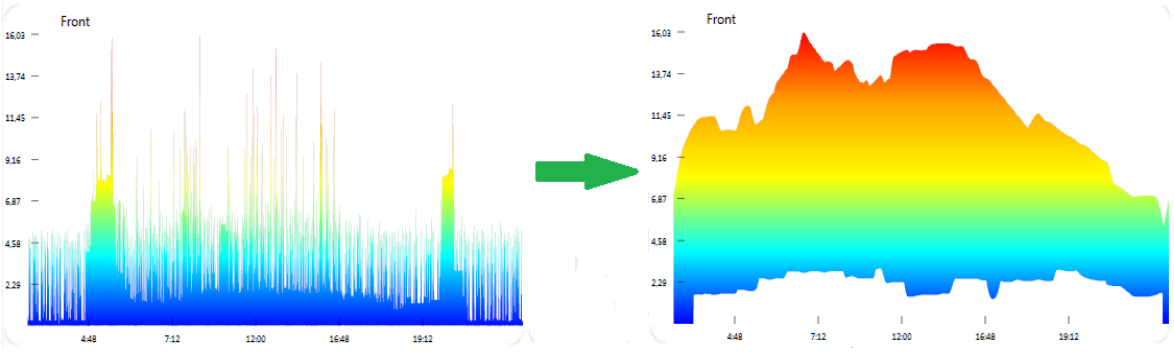
Interpolační model dat

Druhým použitým typem modelu dat je model interpolační. Některé měřené veličiny se vyznačují vysokou mírou proměnlivosti v krátkých časových intervalech (elektrický proud a výkon). Použití interpolace tuto vlastnost do jisté míry eliminuje.

V aplikaci byl tedy využit datový model `AkimaSpline` z knihovny KMB Analytics. Program modelu předá jistou podmnožinu dat, které fungují jako řídící body interpolace. Tyto body následně model proloží křivkou *Akima* a funkcí `GetEstimatedValues` je poté možné získat body na této křivce.

Výsledná vizualizace potlačuje krátkotobé změny a zvýrazňuje změny dlouhodobé. Zároveň dochází ke ztrátě přesnosti výsledného grafu oproti originálním datům. Cílem vytvořených vizualizací je však poskytnout globální informace o měřených veličinách, což použití modelu Akima Spline opodstatňuje.

Na obrázku následuje rozdíl mezi grafem s použitým modelem Akima Spline a grafem původních dat:



Původní graf Akima Spline

Obrázek 7-2:Použití interpolačního modelu Akima Spline

8. Závěr

Výsledkem práce je úspěšně vytvořená klientská aplikace pro zpracování a vizualizaci dat měření kvality elektrické energie. Aplikace byla vytvořena v grafickém subsystému Windows Presentation Foundation a ověřuje možnosti jeho vestavného 3D enginu, s jehož pomocí aplikace generuje trojrozměrné vizualizace dat obsažených v rozsáhlé databázi aplikace ENVIS.

Aplikace implementuje dva různé typy konstrukce 3D grafu. Plošná geometrie vytvoří z naměřených hodnot aproximovanou plochu metodou jednoduché triangulace. Metoda se projevila jako užitečná; s minimálními požadavky na výpočetní výkon a operační paměť vykreslí srozumitelný trojrozměrný graf.

Druhá metoda konstrukce (sloupcová geometrie) se příliš neosvědčila. Každá hodnota je reprezentována kvádrem v 3D prostoru; na každý takový kvádr je třeba vykreslit celkem dvanáct trojúhelníků. U sloupcové geometrie stačil na tři naměřené hodnoty jen jeden trojúhelník. S rostoucím objemem vizualizovaných dat tedy rapidně rostou i nároky na výpočet a operační paměť systému. Vytvoření grafu může trvat i několik minut i na relativně krátkém rozsahu vybraných dat. Při otáčení grafu vytvořeného pomocí sloupcové geometrie v perspektivním pohledu je navíc doba odezvy na akci uživatele tak veliká, že se tato funkce stává v podstatě prakticky nepoužitelnou.

Aplikace statistických modelů na vstupní data do jisté míry redukuje výkonostní nedostatky WPF při použití sloupcové konstrukce grafu; práce s grafem je však stále dost obížná ve srovnání s plošnou geometrií. Z tohoto důvodu bych při dalším vývoji vizualizační aplikace volil jiný grafický engine – SlimDX nebo Microsoft XNA.

WPF se však osvědčilo v místech, kde bylo třeba kombinovat dvourozměrnou grafiku s grafikou trojrozměrnou. Například v grafech vytvořených pomocí ortogonálních projekcí na 3D objekt byly kóty grafu vytvořeny pomocí knihovny WPF pro 2D grafiku. Při použití externího 3D enginu v kombinaci s WPF vyžaduje aplikace vymezit prostor klientské části okna výhradně pro 3D vykreslování. Do tohoto prostoru nesmí zasahovat žádná komponenta subsystému WPF. V případě použití vestavného 3D enginu WPF však bylo možné vložit transparentní panel Canvas (komponenta WPF na vykreslování 2D grafiky) nad prostor určený 3D vykreslování a do toho panelu vkládat popisky os grafu.

Dodatky

Použitá literatura

[1] Petzold Charles: Mistrovství ve Windows Presentation Foundation. 1. vydání, Brno, Computer Press, a. s., 2008. 928 str. ISBN 978-80-251-2141-2.

[2] Petzold Charles: 3D Programming for Windows: Three-Dimensional Graphics Programming for the Windows Presentation Foundation. 1. vydání, Redmond, Microsoft Press, 2007. 448 str. ISBN 978-0-7356-2394-1.

[3] Mehta Vijay P.: Pro LINQ Object Relational Mapping with C# 2008. 1. vydání, Berkley, Apress, 2008. 408 str. ISBN 978-1-59059-965-5

[4] Lerman Julia: Programming Entity Framework, Second Edition. 2. vydání, Sebastopol, O'Reilly Media, Inc., 2010. 912 str. ISBN 978-0-596-80728-3

[5] MacDonald Matthew: Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5, 2. vydání, Berkley, Apress, 2008. 1000 str. ISBN 978-1-59059-955-6

[6] Xu Jack: Practical WPF Charts and Graphics. 1. vydání, New York, Apress, 2009. 712 str. ISBN 978-1-4302-2482-2

[7] Perspective %28graphical%29. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 26.11. 2003, last modified on 12.5. 2011 [cit. 2011-05-18]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Perspective_%28graphical%29>.

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů